Paula Marin

Yleisöverkkopelin teknologiat ja toteutus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Mediatekniikan koulutusohjelma

Insinöörityö

6.5.2016

Metropolia

Tekijä Otsikko Sivumäärä Aika	Paula Marin Yleisöverkkopelin teknologiat ja toteutus 36 sivua 6.5.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Mediatekniikka
Suuntautumisvaihtoehto	Digitaalinen media
Ohjaajat	Teknologiajohtaja Tuomas Paavola Lehtori Ilkka Kylmäniemi

Insinöörityön tarkoituksena oli toteuttaa yleisöverkkopeli, johon tapahtuman yleisö voi osallistua käyttämällä omia päätelaitteitaan. Peli koostuu kolmesta osa-alueesta: tapahtuman näytöillä ja televisiolähetyksessä esitettävästä pelistä, palvelintoteutuksesta ja yleisön käyttämästä web-sovelluksesta. Pelin tulokset päivittyvät kaikissa laitteissa reaaliaikaisesti.

Yleisöverkkopelissä yhdistyy verkkopelaaminen ja yleisöpeli, missä yleisö ottaa osaa johonkin aikaan, paikkaan tai tapahtumaan sidottuun moninpeliin. Yleisön käyttämä sovellus toteutettiin moderneilla web-tekniikoilla, joita ovat HTML5, CSS3 ja JavaScript. Sovelluksen käyttöliittymän toteutukseen käytettiin responsiivista Foundation-sovelluskehystä. Palvelintoteutus koostui node.js-palvelimesta ja Streamr-palvelusta, joka tarjoaa tiedonvälityspalvelua rajapinnan avulla. Tapahtuman näytöillä ja televisiolähetyksessä esitettävä peli toteutettiin Flash-tekniikalla, joskin sen käsittely jätettiin tässä työssä vähemmälle huomiolle.

Työssä pohdittiin yleisöverkkopelin kehityksen lähtökohtia ja vaatimuksia. Vaatimuksista tärkein oli matala osallistumiskynnys ja pelin reaaliaikaisuus. Pelin toiminnan ja toteutuksen kuvauksessa keskityttiin käsittelemään ennen kaikkea pelin rakennetta ja viestintää.

Yleisöverkkopeliä testattiin yleisötapahtumassa sen valmistuttua. Yleisö sai äänestää urheilutapahtuman aikana ottelun parasta kotijoukkueen pelaajaa. Peli toimi ongelmattomasti ja sen kehitystä päätettiin jatkaa. Valmistunutta yleisöverkkopeliä on sittemmin pelattu kahdessa muussa suuressa urheilutapahtumassa.

Avainsanat	yleisöverkkopeli, web-sovellus, flash, node.js

Author Title Number of Pages Date	Paula Marin Development and Technologies of an Online Crowd Game 36 pages 6 April 2016
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructors	Tuomas Paavola, Chief Technology Officer Ilkka Kylmäniemi, Lecturer

The purpose of this bachelor's thesis was to create an online crowd game, for events to which the audience could join using their own devices, for Uplause Oy. The game itself consists of three different components: a game displayed on the screens of the event venue and television broadcasts, server side implementation of the game and the web client provided to the audience. All of these components are running synchronized with one another.

In the thesis the concept for crowd gaming was defined and the techniques used for creation were evaluated. In the end the client software was developed using modern web techniques, HTML5, CSS3 and JavaScript. The user interface was built using responsive Foundation front-end framework whereas the server side implementation was built using a node.js server to provide Streamr - a cloud based stream platform - the stream data for distribution to clients. The component responsible for broadcast at the venue and on television was built using Flash but is left mostly out of the scope of this thesis.

The thesis evaluates various aspects and requirements in development of an online crowd game. Out of the requirements low participation threshold, fault tolerance and real time operation were deemed the most important. In the description of the functionality and implementation the main focus was in the structure and information flow of the game.

The online crowd game was piloted in a crowd game after it was completed. Crowd was able to vote the best player of the home team during the sports event. The game worked without any problems and its development was decided to continue. The online crowd game has been played since in two different major sports events.

Keywords	online crowd game, web application, flash, node.js

Sisällys

Lyhenteet

DOM	Document Object Model, suomeksi dokumenttiobjektimalli. Kuvaa HTML-dokumentin puurakenteen avulla.
HTTP	Hyper Text Transfer Protocol. Selainten ja WWW-palvelimien käyttämä tiedonsiirtoprotokolla.
IETF	The Internet Engineering Task Force. Internet-protokollien standardoinnista vastaava organisaatio.
JSON	JavaScript Object Notation. Yksinkertainen avoimen standardin tiedostomuoto tiedonvälitykseen.
RFC	Internet Official Protocol Standards. RFC:t ovat asiakirjoja, jotka kuvaavat Internetin erilaisia käytäntöjä.
UES	Uplause Entertainment Software. Uplause Oy:n kehittämä yleisöpelejä suorittava asiakasohjelma.
W3C	World Wide Web Consortium, konsortio, joka ylläpitää ja kehittää WWW:n standardeja. .

1 Johdanto

Insinöörityön tarkoituksena on perehtyä erään yleisöpelin tekniikkoihin ja toteutukseen. Yleisöpelissä tapahtumaan osallistunut yleisö toimii ikään kuin pelin ohjaimena. Työn tavoitteena on toteuttaa verkossa pelattava, ensisijaisesti mobiililaitteille suunnattu yleisöpeli. Työn tavoitteena, itse pelin toteuttamisen lisäksi, on tutkia erilaisia teknisiä vaihtoehtoja pelin toteuttamiseksi ja valita niistä sopivat vaihtoehdot hyödyntäen mahdollisuuksien mukaan jo yrityksen käytössä olevaa tekniikkaa.

Insinöörityön toimeksiantaja on sosiaalisia pelejä yleisötapahtumiin kehittävä Uplause Oy. Vuonna 2009 perustettu Uplause Oy toimii Helsingin Salmisaaressa, ja sen markkinat ovat kansainväliset – yrityksen pelejä on viety jo 19 eri maahan. Uplause Oy:n tuottamia pelejä pelataan pääosin erilaisissa urheilutapahtumissa ja festivaaleilla. Yrityksen tuotteet ovat luonteeltaan joko lyhytkestoisia, mainoskatkon aikana pelattavia yleisön äännehdintään ja elehdintään perustuvia pelejä tai kampanjaluonteisia sosiaalisen median rajapintoja hyödyntäviä sovelluksia.

Insinöörityöraportissa tarkastellaan yleisellä tasolla reaaliaikaisen yleisöverkkopelin kehittämiseen soveltuvia tekniikoita. Yleisöverkkopelin toteutukseen kuuluu kolme erillistä kokonaisuutta: yleisön käyttämä mobiilisovellus, tapahtuman näytöillä esitettävä peli ja pelitietoa välittävä palvelin. Mobiilisovelluksen kannalta merkittävimmät teknologiat ovat modernit web-tekniikat: HTML5, CSS3 ja JavaScript sekä erilaiset JavaScript-kirjastot ja responsiiviset sovelluskehykset. Työssä sivutaan myös lyhyesti Adobe Flashia ja Flash Builder –kehitysympäristöä, joilla toteutetaan tapahtuman näytöillä esitettävä peli. Lisäksi työssä käsitellään Node.js-sovellusalustaa, jonka avulla voidaan toteuttaa palvelinpuoli JavaScript-ohjelmointikielellä.

Luvussa 4 keskitytään insinöörityön osana tehtävään peliin sen toteutuksen ja toiminnan näkökulmasta. Yleisöverkkopelin toteutuksessa keskeistä on se, miten pelin sisäinen tiedonkäsittely on ratkaistu, jotta monen pelaajan verkkopeli onnistuu suurissa yleisötapahtumissa. Tällaisten mobiiliverkkoyhteyksiä hyödyntävien moninpelien pelaaminen ei ole ollut mahdollista ennen modernien mobiiliverkkoteknologioiden kehittymistä ja tukiverkkojen yleistymistä.

Osa toteutettavasta sovelluksesta pohjautuu yrityksen aiempiin peleihin ja tekniikkoihin, mutta palvelimen ja web-sovelluksen toteutus aloitettaan tyhjästä. Toteutan pelikokonaisuuden pääasiassa itse, mutta kehitystyössä minua auttavat kokeneemmat sovelluskehittäjät tarpeen tullen.

2 Mobiililaitteille suunnatun yleisöverkkopelin teknologiat

Uplause Oy on ensimmäisenä yrityksenä maailmassa tuottanut aidosti interaktiivisia yleisön ohjaamia pelejä maailmanlaajuisille markkinoille. Yrityksen tuotteita kutsutaan nimellä yleisöpeli (crowd game), joka ei ole yleisesti vakiintunut termi, mutta kuvastaa yrityksen tuotteita varsin hyvin. Langatonta verkkoyhteyttä hyödyntävän yleisöpelin komponentit rakentuvat yrityksen asiakasohjelmasta, jolla ohjataan ja näytetään pelin kulkua tapahtuman näytöillä, mobiilisovelluksesta, jonka avulla pelaajat voivat osallistua yleisöpeliin, ja palvelimesta, jonka tehtävänä on jakaa pelitietoa mobiilisovelluksen ja peliä kontrolloivan asiakasohjelman välillä. Tässä luvussa määritellään tarkemmin yleisöverkkopelin käsite ja pohditaan verkossa pelattavan yleisöpelin tekniikoiden valintaa.

2.1 Yleisöverkkopelin määritelmä

Verkkopelillä tarkoitetaan digitaalista peliä, jonka pelaaminen edellyttää jatkuvaa verkkoyhteyttä. Verkkopelaaminen voi tapahtua verkkosivuston lisäksi pelikonsolilla, matkapuhelimella tai vertaisverkon välityksellä. [1.] Vaikka verkkopelejä voi pelata myös itsenäisesti, sosiaalisuus on usein voimakas motiivi pelata verkossa. Verkkopelit perustuvatkin tiedonsiirron tarjoamiin mahdollisuuksiin, jotka näkyvät etenkin pelien tarjoamissa viestintämahdollisuuksissa. Pelaajien välinen vuorovaikutus on monen pelaajan verkkopeleissä keskeisellä sijalla, ja tällaiset pelit tyypillisesti kannustavat tai jopa pakottavat pelaajien väliseen yhteydenpitoon. [2.]

Moninpelin käsite on laaja. Se voidaan määritellä peliksi, johon osallistuu useampia kuin yksi ihmispelaaja. Yleisöpeli on moninpelin yksi muoto, jossa yleisö toimii pelin ohjaimena – yleensä joko äännehtimällä tai elehtimällä. Yleisöpelille on ominaista se, että se on sidonnainen johonkin aikaan, paikkaan tai tapahtumaan, jossa peliä pelataan. Yleisö tai yksittäinen pelaaja ei kontrolloi pelin alkamista tai päättymistä, vaan

siitä vastaa esimerkiksi tapahtumaorganisaatio. Yleisön tehtäväksi jää ainoastaan peliin osallistuminen. Yleisöpelin tarkoitus on tarjota elämyksiä, lisätä yhteenkuuluvuuden tunnetta sekä aktivoida ja osallistaa yleisöä (kuva 1). Yleisöpelit tarjoavat myös tapahtuman sponsoreille interaktiivisen ja tehokkaan median markkinointiin.

Kuva 1. Vuonna 2014 X-Games tapahtumassa pelattiin Uplause Oy:n yleisöpeliä, jossa yleisön ääni kontrolloi ralliauton vauhtia [3].

Tämän insinöörityön tuloksena syntynyt yleisöverkkopeli yhdistää verkkopelaamisen ja yleisöpelin. Yleisö ei osallistu pelaamiseen enää elehtimällä tai äännehtimällä, vaan verkon välityksellä omalla päätelaitteellaan. Tämä ei ole ollut ennen mahdollista, sillä suuren langattoman viestiliikenteen välittämiseen samasta fyysisestä sijainnista on ollut haasteellista ja ennen kaikkea kallista, sillä se vaatisi esimerkiksi ylimääräisten tukiverkkojen pystyttämisestä tapahtuman ajaksi. 3G- ja 4G-verkkojen kehityksen myötä sekä tukiasemien lisäännyttyä etenkin tapahtumapaikkojen läheisyyteen verkossa pelattavien moninpelien tuominen osaksi tapahtumaa on tullut viime vuosien aikana mahdolliseksi.

2.2 Mobiilisovellus

Verkossa pelattavaan yleisöpeliin osallistuminen vaatii laitteen, jolla saa yhteyden internetiin. Mikäli peliin voi osallistua tapahtumassa paikalla olevan yleisön lisäksi kotiyleisö, päätelaite saattaa olla mikä tahansa älypuhelimen ja pöytätietokoneen välillä. Jotta pelikokemus olisi saumaton laitteesta riippumatta, sovellus kannattaa suunnitella

ensin mobiililaitteille. Paras lopputulos syntyy, kun sovellusta skaalataan pienestä ja yksinkertaisesta kohti suurempaa ja kompleksisempaa sovellusta [4.] Tässä luvussa yleisön käyttämää pelisovellusta kutsutaan mobiilisovellukseksi, vaikka käyttö tulee olla mahdollista myös esimerkiksi pöytätietokoneella. Seuraavaksi pohditaan erilaisia mobiilisovelluksen toteuttamiseen soveltuvia tekniikoita, ja perustellaan tässä työssä valitut tekniikat.

Mobiilisovelluksen toteutustavat

Yleisöpelin tarkoitus on aktivoida ja osallistaa yleisöä, jolloin osallistumiskynnyksen tulisi olla mahdollisimman matala. Sovelluksen käytön ja käyttöönoton tulisikin olla mahdollisimman nopeaa ja helppoa, siispä pelin toteuttamistapa on mietittävä tarkkaan. Mobiilaittelle suunnattua sovellusta kehitettäessä sovelluksen toteuttamistapoja on kolme: natiivi-, hybridi- tai web-sovellus.

Natiivisovellus on laitealustalle erikseen ohjelmoitu sovellus. Eri laitealustat eivät ole yhteensopivia, ja niille on omat erilliset kehitystyökalut. Esimerkiksi Applen iOS-sovellukset ohjelmoidaan Objective-C-ohjelmointikielellä käyttäen Xcode-kehitystyökaluja, kun taas Android-sovelluksia ohjelmoidaan Java-ohjelmointikielellä ja Googlen tarjoamilla kehitystyökaluilla. Natiivisovellusten kehittäminen on haastavaa ja kallista, sillä ensinäkin ennen sovelluksen julkaisua on suoritettava käyttöehtoihin ja sääntöihin liittyvä tarkistusprosessi. Lisäksi natiivisovellusten kehittäminen usealle eri laitteelle vaatii monen eri laitealustan ja teknologian hallitsemista. [5.]

Mobiililaitteiden Internet-selaimet ovat kehittyneet viime vuosina niin paljon, että niille voidaan kehittää nykyaikaisia, mobiilikäyttöön optimoituja web-sovelluksia. Tämän vuoksi on mahdollista, että sama web-sovellus toimii kaikilla laitealustoilla, jolloin niiden kehittäminen on edullisempaa verrattuna natiivisovelluksiin. Web-sovellusten kehittämiseen käytettävät perustekniikat, HTML, CSS ja JavaScript, ovat säilyneet kautta aikojen samana, jolloin näiden tekniikoiden osaajia on markkinoilla paljon enemmän kuin natiivisovellusten kehittäjiä. Lisäksi web-sovellusten julkaisemiseen ei liity jakelukanavien yhteydessä vaadittavaa käyttöehtojen ja sääntöjen hyväksymistä, jolloin myös sovelluksen päivittäminen on nopeaa ja helppoa. Web-sovelluksilta puuttuu natiivisovellusten mahdollisuus hyödyntää kaikkia laitteiden toimintoja, mutta esimerkiksi paikannustietojen saanti onnistuu jo nykyään JavaScript-rajapintojen avulla. [5.]

Hybridimobiilisovellus on sekoitus natiivisovellusta ja web-sovellusta. Hybridisovellus on toteutettu web-sovelluksen tavoin HTML5-tekniikoilla, mutta se suoritetaan puhelimessa natiivisovelluksen tavoin erikseen asennettavana sovelluksena. Sovellus toimii käytännössä kuin selain fullscreen-moodissa, eli ilman selaimen toimintoja, kuten osoitepalkkia ja takaisin-nappia. Toisin kuin web-sovelluksissa, hybridisovellukset voivat hyödyntää natiivisovellusten tavoin kaikkia laitteen toimintoja, tätä varten erikseen kehitettyjen JavaScript rajapintojen avulla. Hybridimobiilisovellukset kehitetään yleensä sitä varten kehitetyllä sovellusalustalla, esimerkiksi Cordovalla (entinen PhoneGap). Cordovan avulla HTML5-tekniikoilla toteutettu sovellus saadaan muutettua natiivisovellukseksi, jonka julkaisuprosessi on sama kuin natiivisovelluksen. Hybridimobiilisovellus yhdistää web-sovelluksen kehittämisen helppouden ja natiivisovellusten yhtenäisen käyttökokemuksen kaikilla eri laitteilla. [5].

Tässä insinöörityössä käsiteltävän yleisöpelin mobiilisovelluksen toteuttamistavaksi valittiin natiivi- tai hybridisovelluksen sijaan web-sovellus. Tähän päätökseen vaikutti erityisesti vaatimus matalasta osallistumiskynnyksestä, kehityskustannusten kalliista hinnasta ja olemassa olevasta osaamisesta. Web-sovellus ei vaadi erillistä asentamista päätelaitteelle, ja peliin pääsee osallistumaan yksinkertaisesti näppäilemällä osoitteen selaimeen. Web-sovelluksen kehitykseen riittää yksinkertaisimmillaan tekstieditori ja ohjelma, jolla voi siirtää tiedostoja palvelimelle, jolloin esimerkiksi maksullisia kehitysympäristöjä ei tarvita. Itselläni oli entuudestaan paljon kokemusta web-kehityksestä, jolloin aikaa ei kulunut web-sovelluksen kehittämiseen käytettävien perustekniikoiden opetteluun.

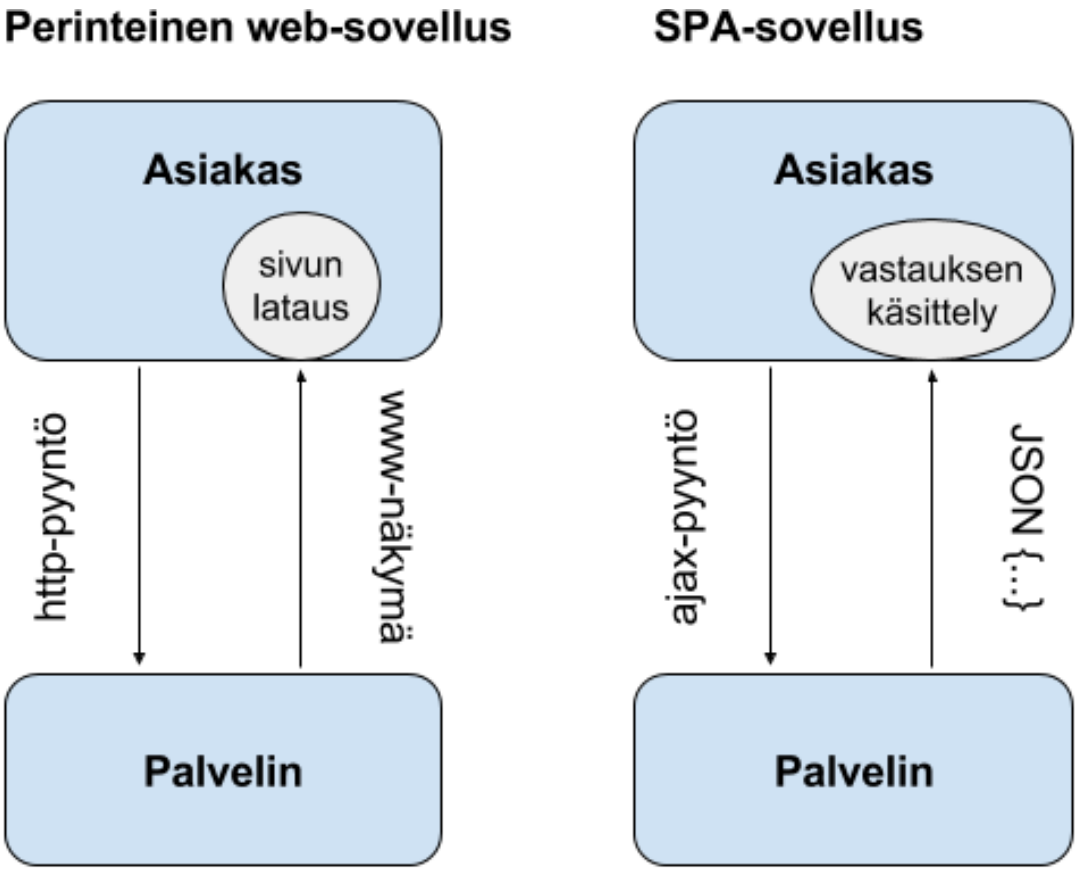
Web-sovelluksen arkkitehtuuri

Dynaaminen web-sovellus, jossa sivun sisältö muuttuu usein, tarvitsee selaimen lisäksi palvelimen, jonka tehtävänä on tarjota resursseja verkon käyttäjille. Kun käyttäjän selainohjelmisto, esimerkiksi Google Chrome, tekee pyynnön verkossa sijaitsevalle palvelimelle, palvelin vastaanottaa pyynnön, käsittelee sen ja rakentaa pyyntöön sopivan vastauksen. Vastaus voi sisältää esimerkiksi web-sivun tai tietyssä muodossa olevaa dataa. [6.]

Dynaamiset web-sovellukset on perinteisesti tehty palvelinpään toteutuksina. Kuten kuvassa 2 esitetään, tällaisessa toteutuksessa palvelinohjelmisto ottaa vastaan http-

pyynnön, prosessoi sen ja palauttaa tämän prosessin tuloksen mukaisen vastauksen, joka on tavallisesti HTML-sivu [7]. Tämä johtaa tilanteeseen, jossa esimerkiksi napin painaminen johtaa koko sivun uudelleenlataamiseen, mikä ei ole nopeaa eikä tehokasta, etenkään mobiililaitetta käytettäessä.

Viime vuosien aikana onkin kehitetty tehokkaampi ratkaisu, SPA-sovellus (Single Page Application), jossa toimintalogiikka on siirretty palvelimelta selaimen suoritettavaksi. Tämä tarkoittaa esimerkiksi sitä, ettei napin painalluksen seurauksena tarvitse ladata koko sivua uudelleen, vaan sivusta voidaan ladata kerrallaan vain osa. Asiakas voi siis lähettää palvelimelle esimerkiksi ajax-pyynnön ja saada vastauksen JSON-muodossa, minkä jälkeen asiakas voi päivittää esimerkiksi pelkän tekstikentän sisällön saapuneen tiedon perusteella. Käytettävyyden näkökulmasta tällä on iso ero verrattuna palvelinpään sovelluksiin. SPA-sovelluksen käyttökokemusta voidaan verrata sovelluksen vasteajan ja käytettävyyden perusteella jopa natiivisovellukseen. [8.]

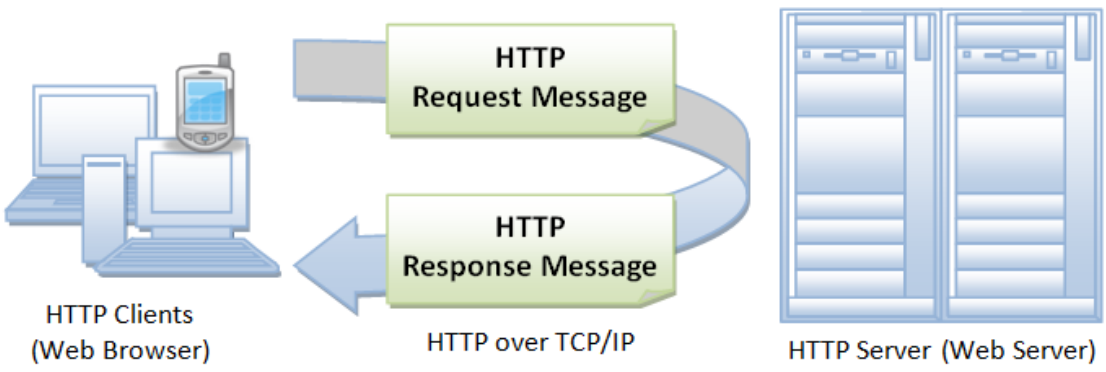


Kuva 2. Dynaamisen sivun palvelupyyntö- ja vastausprosessi. Vasemmalla perinteinen web-sovellus ja oikealla SPA-sovellus.

2.3 Palvelinteknologiat

Palvelin on tietokone, jonka tehtävänä on suorittaa palvelinohjelmistoa. Palvelinohjelmistot tarjoavat palvelimen käyttäjille eli asiakkaille erilaisia resursseja ja palveluita. Tässä insinöörityössä käsiteltävän yleisöverkkopelin toteutukseen on käytetty Amazon (AWS) -pilvipalvelinalustaa mobiilisovelluksen sisällön tallentamiseen ja jakeluun. Lisäksi peliä varten toteutettiin erikseen sovelluspalvelinohjelma, jonka tehtävä on välittää viestejä asiakkaan ja erään toisen palvelun välillä. Tässä luvussa keskitytään jälkimmäisenä mainitun palvelimen teknologiaan.

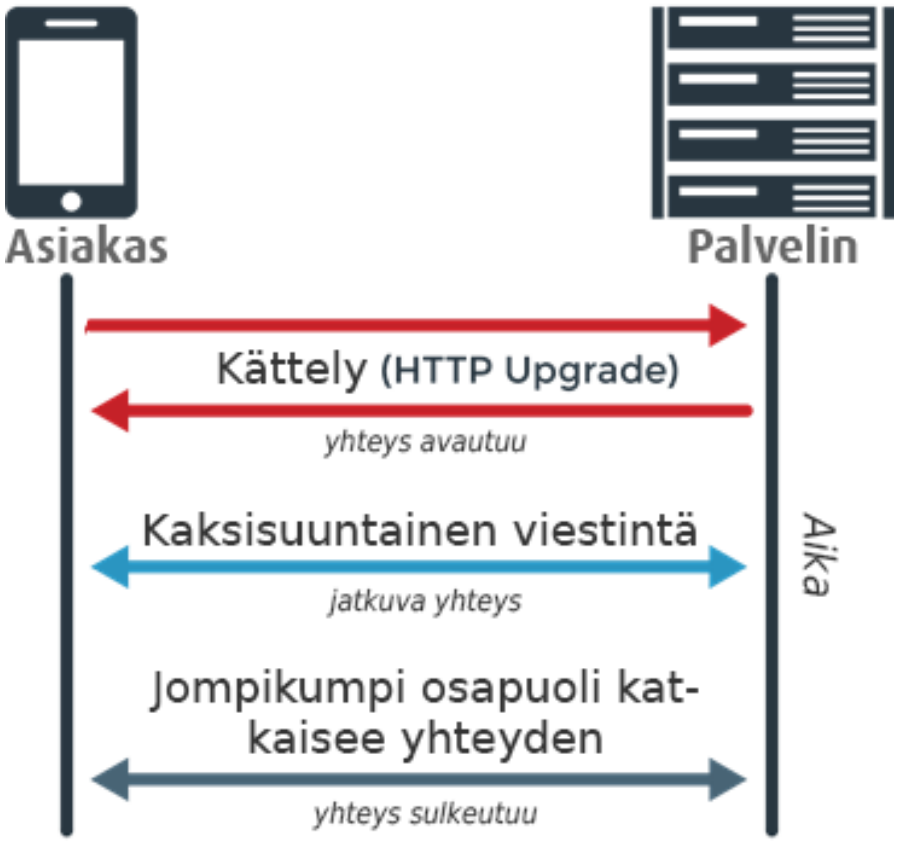
Yleisöverkkopelissä kerätään runsas määrä tietoa lyhyessä ajassa, jolloin palvelimen ja asiakkaan, eli tässä tapauksessa selaimen, on viestittävä mahdollisimman tehokkaasti. Tavallisesti tiedonsiirto tapahtuu web-sovelluksen ja palvelimen välillä HTTP-protokollan avulla, joka perustuu asiakkaan palvelimelle lähettämään pyyntöön ja palvelimen pyyntöä vastaan lähettämään viestiin (kuva 3) [9]. Tällainen protokolla ei kuitenkaan ole kovin tehokas, mikäli palvelimen tehtävänä on lähettää reaaliaikaista tilannetietoa kaikille peliin liittyneille asiakkaille. Tällaiseen viestintään sopii WebSocket-verkkotekniikka, joka muodostaa selaimen välille pysyvän kaksisuuntaisen yhteyden, minkä jälkeen palvelin voi yhteyden muodostamisen jälkeen lähettää oma-aloitteisesti tietoa selaimelle [10].



Kuva 3. HTTP-protokollassa asiakas lähettää HTTP-pyynnön palvelimelle, joka palauttaa HTTP-vastauksen noudattaen TCP/IP-protokollaa [9].

WebSocket-protokolla on vielä melko tuore, mutta se on ottanut tärkeitä edistysaskelia kohti standardisointia. IETF julkaisi vuonna 2011 RFC-muistion, jossa kuvaillaan WebSockettien toiminta [11]. Lisäksi W3C-järjestö on julkaissut samana vuonna ehdotuksen JavaScript-rajapinnasta, jonka avulla web-sivut voivat käyttää WebSocketia [12].

Protokolla koostuu kahdesta osasta: yhteyden avaamisesta niin sanotulla kättelyllä (handshake), minkä jälkeen selaimen ja palvelimen välille muodostuu jatkuva kaksisuuntainen yhteys (kuva 4). Toisin kuin perinteisessä HTTP-tietoliikenteessä, jossa selaimen pitää kysyä palvelimelta tietoa, WebSocket-yhteyden muodostuttua palvelin voi lähettää oma-aloitteisesti tietoa selaimelle [10.] WebSocket-yhteys pysyy yllä yhteyden avauduttua siihen asti, kunnes jompikumpi, asiakas tai palvelin, sulkee yhteyden. WebSocket-tekniikan avulla voidaan siirtää tietoa monessa eri muodossa, esimerkiksi binääridataa, tekstiä ja strukturoitua dataa JSON-muodossa [13].



Kuva 4. WebSocket-protokollan vaiheet [muokattu ja suomennettu lähteestä 14].

Ennen WebSocket-tekniikan kehitystä pollaus (polling) ja streamaus (streaming) olivat ainoat varteenotettavat vaihtoehdot tehokkaaseen reaaliaikaiseen viestintään. Pollaus eli kiertokysely on synkroninen (eli samanaikaisuuteen perustuva) tekniikka, jossa asiakas tekee toistuvasti peräkkäisiä pyyntöjä palvelimelle. Palvelin vastaa pyyntöön joko tiedolla tai asianmukaisella varoitusviestillä. Vaikka pollaus on niin sanotusti varma tekniikka, se kuluttaa paljon resursseja. Streamauksessa asiakas lähettää palvelimelle pyynnön, minkä jälkeen palvelin ylläpitää asiakkaaseen jatkuvaa yhteyttä toimittaen tietoa aina tarvittaessa. Streamaus on paljon käytetty tekniikka etenkin median suoratoistoon verkossa, mutta streamauksessa vaadittavien HTTP-otsakkeiden takia siirrettävien tiedostojen koko on melko suuri ja saattaa siten aiheuttaa viivettä. [15, s. 8.]

3 Käytetyt tekniikat

3.1 Adobe Flash – ja Flash Builder –kehitysympäristöt

Adobe Flash, tai lyhyemmin Flash, on Adobe Systemsin alun perin web-animaatioiden kehitykseen tarkoitettu työkalu, joka on vuosien saatossa kypsynyt interaktiivisesta animaatioformaatista kokonaiseksi sovellusalustaksi ja kehitysympäristöksi. Flashin avulla voidaan julkaista multimediasisältöä eri alustoille, kuten verkkosivustoille ja mobiililaitteille. Flashin projektitiedosto on FLA-muodossa, josta voidaan julkaista SWF-muoto. Vuonna 2016 Flash julkaistiin uudelleen ja nimettiin Animate CC:ksi.

Pelikehityksessä Flashin tehtävänä on hallinnoida vektori- ja rasterigrafiikkaa. Flash mahdollistaa myös grafiikan lisäksi audioresurssien tuomisen suoraan kehitysympäristöön. Interaktiiviset pelit vaativat grafiikan hallinnan lisäksi myös ohjelmointia. Flash-ympäristössä käytettävä ActionScript on olio-ohjelmointiin soveltuva ohjelmointikieli, jonka avulla Flashissa luodut audiovisuaaliset resurssit voidaan muuttaa interaktiivisiksi pelin komponenteiksi.

Adobe Flash Builder on Eclipsen, avoimeen lähdekoodiin perustuvan kehitysympäristön, pohjalta kehitetty ohjelmointiympäristö (IDE), jonka avulla voidaan kehittää pelejä, web-sovelluksia ja järjestelmäriippumattomia työpöytäsovelluksia etenkin Flash-alustalle. Flash Builderissa on työkalut muun muassa MXML- ja ActionScript-

sovellusten editoimiseen, ohjelmakoodin kääntämiseen suoritettavaksi ohjelmaksi ja sovelluksen testaukseen.

3.2 Modernit web-tekniikat

Web-sovellusten tekniikat ovat kehittyneet runsaasti viime vuosina. Nykyään web-sovellusten oletetaan toimivan saumattomasti erilaisilla päätelaitteilla niiden käyttöjärjestelmästä tai fyysisestä koosta riippumatta. Tämän haasteen ratkaisemiseksi on syntynyt paljon erilaisia web-sovellusten kehittämiseen sopivia sovelluskehyksiä, jotka mahdollistavat responsiivisen eli laitekohtaisesti mukautuvan käyttökokemuksen niin mobiililaitteilla kuin tietokoneillakin. Modernien verkkosivujen rakennukseen käytettävät tekniikat ovat HTML5, CSS3 ja JavaScript sekä erilaiset JavaScript-kirjastot ja sovelluskehykset.

HTML5

HTML (HyperText Markup Language) on verkkosivujen kuvauskieli, jonka avulla verkkosivulle luodaan runko. Se määrittelee verkkosivun sisällön, mutta ei toiminnallisuutta. HTML-dokumentti on yksinkertaisesti tekstidokumentti, jonka voi luoda millä tahansa tekstieditorilla, ja se koostuu elementeistä tagien sisällä. Tagit ovat tekstiä <- ja >-merkkien sisällä (kuva 5). Näiden elementtien avulla verkkosivulle luodaan runko, ja siihen voidaan asettaa hyperlinkkejä, kuvia ja mediasisältöjä.

```
<!DOCTYPE html>
<html>
 <head>
  <title>Sivuston otsikko</title>
 </head>
 <body>
  <h1>Otsikko</h1>
  <p>Tämä on esimerkki kappale. <a href="osoite.html">Linkki</a>
  </p>
  <!-- Kommentti -->
 </body>
</html>
```

Kuva 5. Esimerkki HTML-dokumentista.

HTML5 voi tarkoittaa pelkästään vuonna 2014 julkaistua spesifikaatiota [16], mutta se viittaa usein yleisesti moderneihin web-tekniikkoihin [17]. Tärkeimpiä HTML5:n mukanaan tuomia uudistuksia ovat seuraavat:

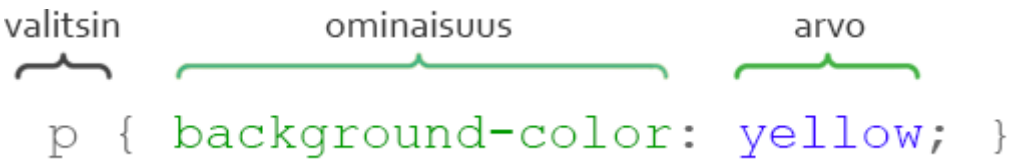
- Semanttiset elementit eli tagit, joiden nimet vastaavat elementin sisältöä, kuvailevat sivun sisältöä tarkemmin. Esimerkiksi <header>-tagi sisältää sivun otsakkeen.
- HTML5 WebSocketit mahdollistavat uuden tavan kommunikoida palvelimen kanssa.
- Verkkosivut voivat tallentaa tietoa selaimeen, mikä mahdollistaa selailun ilman verkkoyhteyttä.
- Uusien multimediaelementtien avulla verkkosivulle voidaan istuttaa videoita ja audiota ilman erillisiä selainlisäosia.
- 2D- ja 3D-grafiikan esitykseen tuotujen parannusten myötä esimerkiksi 3D-grafiikan esittäminen on mahdollista selaimessa ilman selainlisäosia.
- Suorituskyky- ja integraatioparannukset helpottavat verkkosivujen optimointia. Verkkosivu voi esimerkiksi muokata selaimen historiaa ja ottaa koko näytön tilan käyttöönsä.
- Verkkosivu voi käyttää laitteen ominaisuuksia, kuten paikannusta tai kameraa.
- CSS3:n myötä verkkosivujen tyylimäärittelyyn on entistä enemmän työkaluja. [18.]

CSS3

CSS (Cascading Style Sheets) on tyyliohjeiden laji, jolla kuvaillaan, miltä HTML- tai XML-elementtien tulisi näyttää eri medioilla. CSS:llä voidaan siis määritellä koko verkkosivun tyyli. CSS on W3C:n standardoima, ja sitä kehitetään vaiheittain. Tällä hetkellä W3C suosittelee käyttämään CSS2.1-versiota, joskin CSS3 kehittyy nopeasti kohti standardointia. CSS3:n kehitys on pilkottu osiin, ja useimmat modernit selaimet tukevat niistä suosituksiksi edenneitä moduuleita. CSS3:n odotetuimpia uudistuksia ovat muun muassa pyöristetyt reunat, varjot, gradientit ja animaatiot. [19.]

Kuvassa 6 esitetään, miten kappaleen taustaväri voidaan määritellä keltaiseksi. Kappale (paragraph) valitaan <p>-tagia vastaavalla valitsimella p, jonka taustavärin ominai-

suudeksi (background-color) annetaan arvo keltainen (yellow). Tämän syntaksin avulla lähes jokaiselle HTML-elementille voidaan antaa erilaisia tyylimääritelmiä.



Kuva 6. Esimerkki kappaleen taustan värjäämisestä keltaiseksi CSS-tyylimäärittelyn avulla.

JavaScript

JavaScript on kevyt, selaimen tulkitsema dynaaminen komentosarjakieli, jota ylivoimainen enemmistö web-sovelluksista käyttää sivun toiminnallisuuden toteuttamiseen [20, s. 1]. JavaScript perustuu prototyyppeihin, ja sillä on täysi tuki funktionaaliseen oliopohjaiseen ohjelmointiin. JavaScript perustuu ECMAScript-standardiin, jonka versiota 5.1 tukevat kaikki modernit selaimet [21]. Uusin versio ECMAScript 6 (ES6) on julkaistu vuonna 2015, mutta sen käyttö ilman kääntäjää ei ole suositeltavaa, sillä etenkään mobiiliselaimet eivät tue ES6-standardia riittävällä tasolla [22].

JavaScript voi toimia sekä selain- että palvelinympäristössä. Selainympäristössä JavaScriptilla on työkalut selaimen DOM:n muokkaamiseen ja käyttäjän interaktion käsittelyyn. Palvelinympäristössä JavaScript pystyy suorittamaan palvelimen toimintoja, ja se voi kommunikoida esimerkiksi tietokannan kanssa. Tässä insinöörityössä JavaScriptia on käytetty sekä selain- että palvelinympäristössä.

Selainympäristössä käytettynä JavaScriptiä voi kirjoittaa suoraan HTML-sivuun <body>- ja <head>-tagien sisälle tai sitä voidaan kirjoittaa erilliseen ulkoiseen tiedostoon (kuva 7). Ulkoiseen tiedostoon kirjoitettu JavaScript helpottaa koodin lukemista ja ylläpitoa ja nopeuttaa selaimen toimintaa, sillä selain voi tallentaa ulkoisen JavaScript-tiedoston välimuistiin, jolloin sitä ei tarvitse ladata aina uudelleen sivun päivittyessä. Palvelinympäristön JavaScriptistä kerrotaan lisää luvussa 3.3.

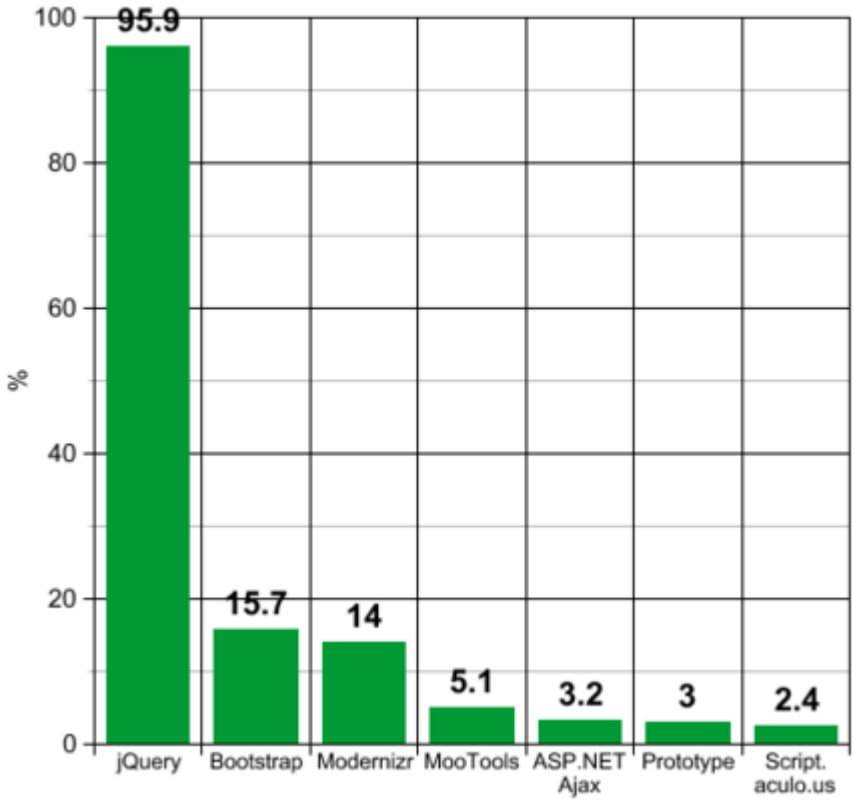
```
<!doctype html>

<head>
    <title> Otsikko </title>

    <!-- Tämä on kommentti -->
    <!-- Linkki ulkoiseen CSS-tiedostoon -->
    <link rel="stylesheet" href="tyylitiedosto.css" media="all">

    <!-- Linkki ulkoiseen JavaScript-tiedostoon -->
    <script type="text/javascript" src="javascript.js"></script>

</head>
<body>

    <!-- Tämän painikkeen (button) onclick-attribuuttiin on lisätty
    viittaus JavaScript-funktioon. Painiketta klikattaessa kutsutaan
    annettua funktiota, joka on määritelty esimerkiksi ulkoisessa
    JavaScript tiedostossa. -->
    <button type="button" onclick="funktio()">Paina tästä</button>

    <!-- JavaScriptiin voidaan viitata head-tagin lisäksi myös
    esimerkiksi body-tagin lopussa. Sama HTML-tiedosto voi sisältää
    useita linkkejä eri JavaScript tiedostoihin. -->
    <script type="text/javascript" src="javascript2.js"></script>

</body>
</html>
```

Kuva 7. JavaScript voidaan upottaa script-tagien avulla HTML-sivuun.

JavaScript-ohjelmointia helpottamaan on kehitetty runsaasti erilaisia JavaScript-kirjastoja. Tietotekniikan yhteydessä kirjastolla tarkoitetaan kokoelmaa luokkia ja ohjelmia, joita voidaan liittää kehitettävän sovelluksen avuksi tarpeen tullen. Kirjastojen tehtävänä on ratkaista yleisiä ohjelmointitehtäviä mahdollisimman tehokkaasti, jolloin JavaScriptin kirjoittaminen selkeytyy ja helpottuu huomattavasti.

Kuvassa 8 kuvataan W3:n maaliskuussa vuonna 2016 tehty otanta JavaScript-kirjastojen käytöstä eri verkkosivuilla. W3:n seuraamista verkkosivustoista 27,5 % ei käytä mitään taulukossa esiintyvää JavaScript-kirjastoa. Taulukossa esiintyvät prosenttiluvut perustuvat vain niihin verkkosivuihin, joissa käytetään jotain W3:n listaamaa JavaScript-kirjastoa. jQuerya, ylivoimaisesti suosituinta JavaScript-kirjastoa, käyttää kai-

kista verkkosivustoista yhteensä 69,5 %, jolloin sen osuus kaikista taulukon JavaScript-kirjastoa käyttävistä sivustoista on 95,5 % [23].



Kuva 8. JavaScript-kirjastojen käyttö verkkosivuilla perustuen W3:n maaliskuussa 2016 tehtyyn otantaan verkkosivuista, jotka käyttävät vähintään yhtä W3:n seuraamaa JavaScript-kirjastoa [23].

jQuery on John Resignin vuonna 2006 julkaisema avoimen lähdekoodin JavaScript-kirjasto, jonka tarkoitus on helpottaa HTML-dokumenttien ja JavaScriptin muokkaamista ja korjata selainten välisiä ongelmia. Tällä hetkellä yli 53 miljoonaa aktiivista verkkosivustoa käyttää jQuerya. [24.]

Kuvassa 9 esitetään luokka-attribuutin (class) lisääminen div-elementtiin sekä JavaScriptin että jQueryn avulla. Kuten kuvasta voidaan havaita, jQuerylla attribuutin lisääminen onnistuu pienemmällä määrällä tekstiä. JQuery etsii elementin muuttujaan *id*-attribuutin perusteella $( )-funktion avulla ja lisää siihen luokan jQueryn *addClass*-funktiolla.

```
// Div-elementti, jonka sisällä on tekstiä.
<div id="elementti"> Tekstiä </div>

// Luokka-attribuutin lisääminen div-elementtiin
// 1. JavaScriptin avulla:
document.getElementById('elementti').setAttribute('class', 'className');

// 2. jQueryn avulla:
$("#elementti").addClass("className");

// Lopputulos:
<div class="className"> Div-elementti </div>
```

Kuva 9. Luokka-attribuutin lisääminen div-elementtiin JavaScriptin ja jQueryn avulla.

Responsiiviset HTML5-sovelluskehykset

Sovelluskehys on kirjaston tapaan eräänlainen sovelluskehittäjän työkalupakki, joka ratkaisee sovelluskehitykseen liittyviä yleisiä ongelmia tarjoamalla valmiita sovelluskomponentteja ja käytäntöjä toteutuksen perustaksi. Kirjasto ja sovelluskehys ovat käsitteinä osittain päällekkäisiä, ja niiden tavoite on sama – helpottaa ja nopeuttaa sovelluskehitystä. Sovelluskehykset eroavat kirjastoista kuitenkin siten, että ne tarjoavat sovellukselle ikään kuin valmiin rungon ja strategian ongelmanratkaisuun. Kirjastot taas keskittyvät usein ratkaisemaan vain jotain tiettyä sovelluskehityksen ongelmaa tai osa-aluetta.

Responsiivisuudella tarkoitetaan sitä, että sovellus mukautuu sitä käyttävään laitteeseen. Tällä tarkoitetaan web-kehityksessä sitä, että CSS3-mediakyselyiden avulla voidaan selvittää selainikkunan koko, alusta ja orientaatio, jolloin web-sivun sisältö voidaan skaalata ja mukauttaa selainikkunaan sopivaksi. Responsiiviset ulkoasut rakennetaan useimmiten ruudukon (grid) avulla, minkä koko määritellään suhteellisilla arvoilla. Modernit verkkosivut käyttävätkin apunaan erilaisia responsiivisia web-sovelluskehyksiä ja kirjastoja, jotka tarjoavat ratkaisun web-sivun sisällön mukauttamiseen erikokoisille laitteille.

Suosittuja responsiivisia HTML5-sovelluskehyksiä ovat muun muassa Bootstrap, Foundation, SoloBuzz, HTML5 Boilerplate, UIkit ja Skeleton. Tässä insinöörityössä käytetty Foundation on ZURB-nimisen yhtiön vuonna 2011 julkaisema responsiivinen sovelluskehys. Foundation tarjoaa kaiken tarvittavan verkkosivun käyttöliittymän luon-

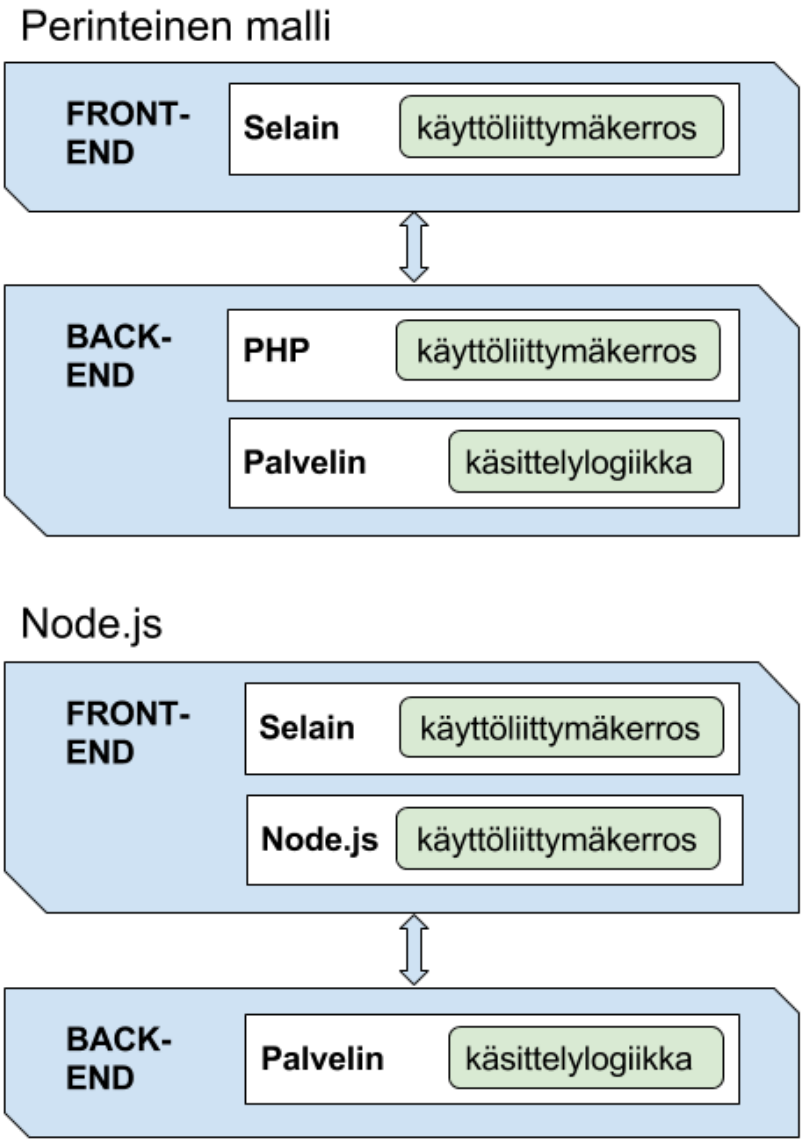
tiin. Ulkoasu rakentuu joustavan ruudukon avulla, joka skaalautuu ja asemoituu automaattisesti selainikkunan kokoon. Foundationilla on myös runsaasti erilaisia valmiita komponentteja, kuten lomakkeita, latauspalkkeja ja videosoittimia, joiden avulla verkkosivuston kehitys helpottuu ja nopeutuu.

3.3 Node.js

Node.js on Ryan Dahlin vuonna 2009 kehittämä avoimen lähdekoodin järjestelmäriippumaton ajoympäristö, jolla voidaan kehittää palvelinohjelma JavaScript-ohjelmointikielellä [25]. Node.js käyttää Googlen kehittämää V8 JavaScript-virtuaalikonetta, ja se soveltuu sen ansiosta raskaaseen käyttöön paljon dataa hyödyntävien ja tosiaikaisten sovellusten kehitykseen [26]. Node.js:n tekniset vaatimukset ovat melko niukat, ja sitä voidaan ylläpitää kevyilläkin pilvialustoilla [27].

Node.js-palvelinsovellukset ohjelmoidaan lähtökohtaisesti asynkronisesti, minkä ansiosta se pystyy ylläpitämään useita samanaikaisia yhteyksiä yllä ilman, että sovellus tukkeutuu. Asynkronisuudella tarkoitetaan sitä, että eri toimintoja voidaan suorittaa samanaikaisesti, jolloin yhden prosessin suorittaminen ei estä toisen prosessin alkamista. Node.js-ympäristössä asynkronisuus mahdollistaa esimerkiksi sen, että HTTP-pyyntöihin voidaan kytkeä useita asynkronisia palvelukutsuja. [26; 27.] Lisäksi toisin kuin monissa muissa palvelintekniikoissa, tapahtumasilmukka on osa Node.js:n rakennetta eikä se ole erillisenä kirjastona. Tämän ansiosta yksittäisen tapahtuman aloittaminen ei pysäytä sovellusten muita toimintoja. [28.]

Web-sovelluskehityksen näkökulmasta Node.js on tuonut ison muutoksen perinteiseen kaksikerroksiseen malliin, johon kuuluvat front-end- ja back-end-kerrokset. Front-end edustaa web-sovelluksesta sitä osaa, johon kuuluu sovelluksen käyttöliittymä, ulkoasu ja vuorovaikutuksen mahdollistama toimintalogiikka. Perinteisessä mallissa back-end sisältää palvelinpään UI-kerroksen, varsinaisen käsittelylogiikan ja tietokannan. Node.js vaihtaa käyttöliittymäkerroksen palvelinlogiikan front-end-kehityksen piiriin, jolloin sovelluksen käyttöliittymäkerroksesta vastaava kehittäjä voi toteuttaa myös siihen vaadittavan palvelinlogiikan – ja kaiken lisäksi samalla ohjelmointikielellä (kuva 10). [29.]



Kuva 10. Web-sovelluksen kaksikerroksinen malli perinteisesti ja node.js:llä toteutettuna.

Socket.io

Socket.io on JavaScript-moduuli, joka mahdollistaa kaksisuuntaisen katkeamattoman yhteyden luomisen Node.js-palvelimen ja asiakkaan välille. Jotta yhteys voidaan muodostaa, sekä asiakkaalla että Node.js palvelimella on oltava socket.io-kirjasto käytössään. Palvelimelle socket.io voidaan asentaa Node.js:n oman paketinhallintatyökalun npm:n avulla. Kuvassa 11 esitetään yksinkertainen socket.io:ta käyttävä Node.js-palvelin, joka kuuntelee porttia numero 3000.

```
var server = require('http').createServer();
var io = require('socket.io')(server);
io.on('connection', function(socket){
  socket.on('event', function(data){});
  socket.on('disconnect', function(){});
});
server.listen(3000);
```

Kuva 11. Yksinkertainen socket.io:ta käyttävä Node.js-palvelin [30].

4 Yleisöverkkopeli

Tämä luku on salattu.

5 Yhteenveto

Insinöörityönä tehtiin yleisöverkkopeli Uplause Oy:lle, joka on toimittanut yleisön äännehdintään ja elehdintään perustuvia yleisöpelejä pääasiassa erilaisiin urheilutapahtumiin ja festivaaleille. Yleisöverkkopeli on käsitteenä uusi: sillä tarkoitetaan yleisötapahtuman osana pelattavaa peliä, jossa yleisö osallistuu peliin käyttämällä omia päätelaitteitaan langattoman verkon välityksellä. Reaaliaikaisten yleisöverkkopelien pelaaminen on tullut mahdolliseksi vasta viime vuosien aikana langattomien verkkojen kehityttyä ja tukiasemien lisäännyttyä etenkin yleisten tapahtumapaikkojen läheisyyteen.

Yleisöverkkopeli koostuu kolmesta eri osa-alueesta: tapahtuman näytöillä esitettävästä yrityksen omalla asiakasohjelmalla (UES) kontrolloitavasta pelistä, palvelintoteutuksesta ja yleisön käyttämästä pelisovelluksesta. Insinöörityössä perehdyttiin näiden osa-alueiden toteuttamiseen käytettyihin tekniikkoihin. UES:llä kontrolloitava peli toteutettiin samoja tapoja ja tekniikoita noudattaen kuin yrityksen muutkin yleisöpelituotteet, joten sen käsittely jätettiin tässä työssä vähemmälle.

Yleisön käyttämän pelisovelluksen mahdollisista toteutustavoista valittiin web-sovellus. Web-sovelluksen hyötyinä on niiden kehittämisen helppous: samaa sovellusta voidaan käyttää kaikilla selaimen omaavilla päätelaitteilla laitealustasta riippumatta. Lisäksi web-sovellus ei vaadi erikseen asennettavaa sovellusta, vaan peliin osallistuakseen tarvitsee näppäillä vain pelin osoite selaimen osoitepalkkiin. Web-sovelluksen kehitykseen ei myöskään vaadita uusien ohjelmointikielien opettelua, mikäli web-kehityksen perustekniikat, HTML, CSS ja JavaScript, ovat ennestään hallussa. Web-sovelluksen kehitykseen käytettiin myös responsiivista HTML5 sovelluskehystä, Foundationia, jonka avulla web-sovelluksen käyttöliittymä saadaan mukautumaan sovellusta käyttävään päätelaitteeseen sopivaksi.

Palvelintoteutus koostui kahdesta osasta: FlashIOBridgestä ja Streamr-palvelusta, joka tarjoaa reaaliaikaista viestinvälitystä rajapinnan avulla. FlashIOBridge on Node.js-tekniikalla yleisöverkkopeliä varten toteutettu palvelinsovellus, joka toimii eräänlaisena tulkkina UES:n ja Streamr-palvelun välillä. Node.js mahdollistaa palvelimen kehityksen JavaScriptilla, samalla ohjelmointikielellä, jolla myös web-sovelluksen toiminnallisuus ohjelmoitiin. Node.js:ään liitetty socket.io-kirjasto taas mahdollistaa WebSocket-tekniikan käytön Node.js-palvelimella.

Toteutetun yleisöverkkopelin kuvauksessa keskityttiin kuvaamaan pelin toimintaa ja toteutusta. Lisäksi pohdittiin yleisöverkkopelin kehityksen lähtökohtia ja vaatimuksia. Vaatimuksista tärkeimmät olivat pelin reaaliaikaisuus, virhetilanteiden sietokyky ja toimivuus mahdollisimman laajalla laitepohjalla. Toiminnan ja toteutuksen kuvauksessa keskityttiin pelin kannalta kiinnostavimpaan osa-alueeseen eli siihen, miten pelin sisäinen tiedonkulku oli toteutettu.

Yleisöverkkopeliä testattiin yleisötapahtumassa sen valmistuttua. Peli esitettiin tapahtuman ja web-sovelluksen lisäksi suorassa televisiolähetyksessä (kuva 20). Yleisö sai äänestää urheilutapahtuman aikana ottelun parasta kotijoukkueen pelaajaa kolmesta vaihtoehdosta, jotka televisiolähetyksen juontajat valitsivat. Ottelun aikana peli toimi ongelmattomasti, ja välitti noin 3 000 ääntä sekä tapahtumapaikalta että kotikatsomoista. Äänestystulokset olivat yhdenmukaiset sekä tapahtuman suurilla näytöillä että web-sovelluksessa. Pelin konsepti todettiin toimivaksi ja kehityskelpoiseksi, vaikka parantamisen varaakin oli. Esimerkiksi äänestysvaihtoehtojen määrä ei saisi olla rajattu, vaan web-sovelluksen tulisi voida esittää kaikki kotijoukkueen pelaajat, joista kolme parasta näytettäisiin tapahtuman suurilla näytöillä ja televisiolähetyksessä.

Kuva 12. Yleisöverkkopeliä kontrolloitiin ohjaamosta käsin.

Pelin kehitystä jatketaan insinöörityön valmistuttua, ja se on herättänyt kiinnostusta eri tahoilla. Itse peliin kehitetään erilaisia pelimoodeja ja asetuksia, jotta peli voidaan hel-

posti mukauttaa tapahtuman luonteeseen sopivaksi. Yleisön käyttämä web-sovellus on mahdollista integroida osaksi natiivisovellusta, jolloin se voi esimerkiksi olla osana jonkin urheilujoukkueen omaa sovellusta, joka suurimmalla osalla faneista olisi jo etukäteen asennettuna mobiililaitteeseensa.

Lähteet

1 Verkkopelit. 2015. Verkkodokumentti. PEGI. <http://www.pegi.info/fi/index/id/203/>. Luettu 24.2.2016.

2 Siitonen, Marko. 2007. Social interaction in online multiplayer communities. Verkkodokumentti. University of Jyväskylä.

3 Uplause. 2016. Verkkodokumentti. Uplause. <http://uplause.com/>. Luettu 25.2.2016.

4 Mobile First Design: Why It's Great and Why It Sucks. Verkkodokumentti. Code My Views. <https://codemyviews.com/blog/mobilefirst>. Luettu 25.3.2016.

5 Vuorinen, Carl. 2014. Kolme tapaa kehittää mobiilisovellus. Verkkodokumentti. <http://w3.fi/kolme-tapaa-kehittaa-mobiilisovellus/>. Julkaistu 21.10.2014. Luettu 28.2.2016.

6 Vihavainen, Arto & Agile Education Research. Web-palvelinohjelmointi. Verkkodokumentti. University of Helsinki. <http://wepa.mooc.fi/index.html#chapter1-1>. Luettu 28.2.2016.

7 Vainionpää, Veijo. 2005. Näkökohtia tehokkaan web-sovelluksen suunnitteluun. Verkkodokumentti. Helsingin ammattikorkeakoulu Stadia. <http://www.metropolia.fi/fileadmin/user_upload/Julkaisutoiminta/Julkaisusarjat/Opinnaytetyot/PDF/Opinnaytetyoet_5_verkkojulkaisu.pdf>.

8 Viskari, Mikko. 2015. Tilaisitko yhden sivun web-sovelluksen (SPA) vai arkkitehtuuriltaan valmiiksi vanhentuneen järjestelmän? Verkkodokumentti. <http://www.eatech.fi/fi/blogi/spa-sovellukset>. 18.11.2015. Luettu 10.3.2016.

9 Hock-Chuan, Chua. 2009. HTTP (HyperText Transfer Protocol). Verkkodokumentti. <https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html>. Last modified 20.10.2009. Luettu 10.3.2016.

10 Sani, Ilari. 2011. Kaksisuuntaista kommunikaatiota ja moninpelejä webissä: WebSocket valmistuu. Verkkodokumentti. Tivi. <http://www.tivi.fi/Arkisto/2011-12-12/Kaksisuuntaista-kommunikaatiota-ja-moninpelej%C3%A4-webiss%C3%A4-WebSocket-valmistuu-3188657.html>. Julkaistu 12.12.2011. Luettu 11.3.2016.

11 Fette, Ian & Melnikov, Alexey. 2011. The WebSocket Protocol. Verkkodokumentti. <https://tools.ietf.org/html/rfc6455>. December 2011. Luettu 10.3.2016.

12 Hickson, Ian. 2011. The WebSocket API. Verkkodokumentti. W3C. <https://www.w3.org/TR/2011/WD-websockets-20110929/>. 29.7.2011. Luettu 10.3.2016.

13 Communications technologies for the Web of Things. 2015. Verkkodokumentti. W3C. <https://www.w3.org/community/wot/wiki/Communications_technologies_for_the_Web_of_Things#Web_Sockets/>. 24.6.2015. Luettu 11.3.2016.

14 Jaitla, Jasdeep. 2015. WebSockets vs REST: Understanding the Difference. Verkkodokumentti. PubNub. <https://www.pubnub.com/blog/2015-01-05-websockets-vs-rest-api-understanding-the-difference/>. January 5, 2015. Luettu 10.3.2016.

15 Pterneas, Vangos. 2013. Getting Started with HTML5 WebSocket Programming. Packt Publishing.

16 HTML5 is a W3C recommendation. 2014. Verkkodokumentti. W3C. <https://www.w3.org/blog/news/archives/4167>. 28 October 2014. Luettu 13.3.2016.

17 Koch, Peter-Paul. 2010. HTML5 means whatever you want it to mean. Verkkodokumentti. Quirksmode. <http://www.quirksmode.org/blog/archives/2010/01/html5_means_wha.html>. Written on 11 January 2010. Luettu 13.3.2016.

18 HTML5. 2009. Verkkodokumentti. Mozilla Developer Network. <https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5>. Last updated Feb 1, 2016. Luettu 13.3.2016.

19 CSS. 2005. Verkkodokumentti. Mozilla Developer Network. <https://developer.mozilla.org/en-US/docs/Web/CSS>. Last updated Jan 26, 2016. Luettu 13.3.2016.

20 Flanagan, David. 2011. JavaScript: The Definitive Guide. Sebastopol, CA: O'Reilly Media.

21 ECMAScript 6 support in Mozilla. 2012. Verkkodokumentti. Mozilla Developer Network. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/New_in_JavaScript/ECMAScript_6_support_in_Mozilla>. Last updated Mar 13, 2016. Luettu 15.3.2016.

22 ECMAScript6 compatibility table. Verkkodokumentti. <https://kangax.github.io/compat-table/es6/>. Luettu 15.3.2016.

23 Usage of JavaScript libraries for websites. 2016. Verkkodokumentti. W3Techs. <http://w3techs.com/technologies/overview/javascript_library/all>. Luettu 15.3.2016.

24 jQuery Usage Statistics. 2016. Verkkodokumentti. BuiltWith. <http://trends.builtwith.com/javascript/jQuery>. Luettu 16.3.2016.

25 Wen, Ben. 2013. 6 things you should know about Node.js. Verkkodokumentti. JavaWorld. <http://www.javaworld.com/article/2079190/scripting-jvm-languages/6-things-you-should-know-about-node-js.html>. Dec 12, 2013. Luettu 28.3.2016.

26 Vänskä, Olli. 2012. Node.js mullistaa ohjelmointia. Verkkodokumentti. TiVi. <http://www.tivi.fi/Uutiset/2012-01-26/Node.js-mullistaa-ohjelmointia-3189519.html>. Päivitetty 26.1.2012. Luettu 28.3.2016.

27 Salonen, Jaakko. 2012. Johdanto JavaScript-sovellusten kehitykseen Node.js:llä. Verkkodokumentti. <http://blite.iki.fi/artikkelit/javascript-nodejs-johdanto/>. Päivitetty 17.9.2012. Luettu 28.3.2016.

28 API DOCS. 2016. Verkkodokumentti. Node.js. <https://nodejs.org/en/docs/>. Luettu 28.3.2016.

29 C. Zakas, Nicholas. 2013. Node.js and the new web front-end. Verkkodokumentti. NCZOnline. <https://www.nczonline.net/blog/2013/10/07/node-js-and-the-new-web-front-end/>. Posted October 7, 2013. Luettu 28.3.2016.

30 socket.io. 2016. Verkkodokumentti. npm. <https://www.npmjs.com/>. Luettu 28.3.2016.

31 Anderson, Shaun. 2016. How Fast Should A Website Load? Verkkodokumentti. <http://www.hobo-web.co.uk/your-website-design-should-load-in-4-seconds/>. Last updated February 13th, 2016. Luettu 20.3.2016.